FAQ on CMT

Index

Providing sourceless documents

I’m always trying to use CMT together with java and I now have problems with the documentation.

What I’d like to do is just generating automatically my documentation using javadoc. Since it is not done in cmt, I tried to write a new fragment to define a new document type : javadoc.

The problem is that the process of generating the doc is not a file to file process : it does not take a .java file to write a .html. It takes a java package name (which has no associated file) and returns many files. This is not possible with the current implementation of document. It seems that adding this feature would lead to write some code into cmt_generator.

Does anyone see another method to generate my documentation ?

Sebastien

In fact what you ask for is already available in CMT, it is documented (although likely crypticly ;-)) as follows :

Since your "document generator" does not make use of any source file per se, it’s likely that you will provide as the fragments the following files (I use as an example the name "javadoc" for your document type):

```
...
make_fragment javadoc_header
make_fragment javadoc -header=javadoc_header
...
```

where javadoc is simply ... empty.

I remind you that here :

- javadoc_header will be instanciated once per document
- javadoc will be instanciated once per source file (thus never if no source file is specified)

Then a document associated with the javadoc type could be specified as follows, using the *unusual* feature I mentioned above

```
document javadoc MyDoc JAVAPACKAGE=MyPackage
```

Here the syntax **JAVAPACKAGE=MyPackage** corresponds to providing user-defined templates in the document's fragments (see the general syntax) every variable name should then correspond within the fragment to constructs like (appearing anywhere in the fragment)

```
...
... ${JAVAPACKAGE} ...
...
```

*(please note the **${..}** syntax with required braces around the variable name)*

You have probably already understood that CMT provides internal predefined similar variable names

the typical example being **CONSTITUENT** which is replaced by the constituent name when the fragment is instanciated.

What I'm talking about here consists in adding any number of user-defined variables which value will be specified on the **document** statement line using any number of **<variable-name>=<value>** constructs.

You will notice that the all-uppercase convention has been selected for CMT-internal variables (eg. **CONSTITUENT**). This is why I have used this convention for this example. This is only a convention, any form is possible here. The only constraint being that it should not overlap with a macro name or an env. var name.

Accessing packages in a sub-directory

I tried to use CMT with sub-packages. How does this work? If I have on cvs a structure like

```
Velo/VeloEvent/..
    /VSicbCnv/..
```

and on disk:

```
Velo/VeloEvent/v3/..
    /VSicbCnv/v3/..
```

*it recognizes the stuff in the use statement, but when I say **cmt show packages** they are not there... Hence, the add-in for devstudio fails.*

Well this is just a subtle effect :

- when you are in the context of say **VeloEvent/v3**, then the path of the current package is implicitly added to the search path, all the used packages that reside in this same root are visible (which explains that the **cmt show uses** works).
- but, when you ask for **cmt show packages**, this is performed outside of any context (mainly because it can be done from any location). Then only packages reachable in the search path (ie. **CMTPATH** or in the **CMT\path** registry set) are visible.

Then in your precise case, you can do one of the following :

1. If you strictly have (as you explained)

   ```
   xxx/Velo/VeloEvent/v3/..
           /VSicbCnv/v3/..
   ```

   then the **CMTPATH** should include **xxx/Velo** in order to make both **VeloEvent** and **VSicbCnv** visible.
2. If **Velo** is itself a package :

   ```
   xxx/Velo/VeloEvent/v3/..
           /VSicbCnv/v3/..
           /v1/...
   ```

   then the **CMTPATH** may simply include **xxx** (although **xxx/Velo** is possible too but in this case, it is redundent). This makes all packages below **Velo** visible.

The difference between 1) and 2) is really the fact that in 2) **Velo** IS a true **CMT** package, whereas, in 1) **Velo** is just a prefix.

For simple efficiency reasons, **CMT** does not scan any arbitrary directory tree depth when it looks for all packages, but only the tree of true **CMT** packages + one level below each package.

Christian Arnault

Last modified: Thu Nov 9 15:16:35 MET 2000